**run\_s\_p.py Utility Documentation**

**File Name**

run\_s\_p.py

**Purpose**

This utility infers whether each contact in an amateur radio contest log (provided as a Pandas DataFrame) was made while "Running" (calling CQ) or "Search & Pounce" (tuning the band). It processes the DataFrame, adds a new 'Run' column with the classification, and the main execution block demonstrates how to read a CSV, use the function, and save the result to a new CSV file.

**Author & Contact**

Mark Bailey, KD4D Contact: kd4d@kd4d.org

**License**

Mozilla Public License, v. 2.0 (<https://www.mozilla.org/MPL/2.0/>)

**Revision History**

All notable changes to this project will be documented in this file. The format is based on "Keep a Changelog" (<https://keepachangelog.com/en/1.0.0/>), and this project aims to adhere to Semantic Versioning (<https://semver.org/>).

**0.9.0-Beta - 2025-07-18**

* Initial Beta release of the Run/S&P Inference utility. This version provides core functionality for analyzing Amateur Radio Contest logfiles in CSV format to classify each contact as either "Running" or "Search & Pounce". It implements a dynamic heuristic based on time windows, QSO rates, and frequency tolerances per band and mode, for each unique operator stream. The utility generates an output CSV file with an added 'Run' column containing the inferred classification.
* Refactored process\_contest\_log\_for\_run\_s\_p to operate directly on a Pandas DataFrame. CSV file reading and writing responsibilities have been moved to the if \_\_name\_\_ == "\_\_main\_\_": block.

**Changed**

* Initial 0.9.0-Beta Release, 2025-07-18

**Fixed**

* (None)

**Removed**

* (None)

**Attribution and Documentation**

This Python implementation is designed to infer operating styles ("Running" vs. "Search & Pounce") from Amateur Radio Contest logfiles. This analysis is crucial for understanding operator strategy and performance during contests.

The core logic for differentiating these operating modes is based on established contest operating practices and statistical patterns observed in QSO rates and frequency usage over time.

This code was entirely written by Google's Gemini AI Assistant and based on Mark Bailey, KD4D's original software. KD4D's software has been used for contest log analysis for over ten years, providing a proven foundation for this utility.

**Usage**

This utility is primarily intended to be imported and used by other Python programs that already have contest log data loaded into a Pandas DataFrame.

**As a Library**

To use this utility as a library:

1. Ensure the run\_s\_p.py file is accessible in your Python environment (e.g., in the same directory as your main script).
2. Import the main function:

Python

from run\_s\_p import process\_contest\_log\_for\_run\_s\_p

1. Prepare your contest log data as a Pandas DataFrame. If your data is in a CSV file, you can load it like this:

Python

import pandas as pd

my\_log\_df = pd.read\_csv("path/to/your\_input\_log.csv", dtype=str)

# Note: `dtype=str` is recommended during initial read to prevent Pandas from

# incorrectly inferring types, as the utility handles type conversion internally.

1. Invoke the process\_contest\_log\_for\_run\_s\_p function with your DataFrame:

Python

processed\_df = process\_contest\_log\_for\_run\_s\_p(

df=my\_log\_df,

# Optional arguments can be passed if your column names differ from defaults or

# you want to customize run detection parameters:

# my\_call\_column='MyStationCall',

# datetime\_column='QSODateTime',

# frequency\_column='FreqMHz', # Example if in MHz

# mode\_column='OpMode',

# band\_column='ContestBand',

# run\_time\_window\_minutes=15, # Default is 10 minutes

# freq\_tolerance\_cw=0.05, # Default is 0.1 kHz

# freq\_tolerance\_ph=1.0 # Default is 0.5 kHz

)

The processed\_df will be a new Pandas DataFrame with all original columns plus the new 'Run' column ('Run' or 'S&P'). The original DataFrame my\_log\_df remains unchanged.

1. You can then save this processed\_df to a new CSV file or use it directly for further in-memory analysis:

Python

# Example of saving to CSV:

output\_csv\_path = "path/to/your\_output\_log\_with\_run.csv"

# Format Datetime column back to string for clean CSV output

datetime\_output\_format = '%Y-%m-%d %H:%M:%S'

df\_for\_output = processed\_df.copy()

df\_for\_output['Datetime'] = df\_for\_output['Datetime'].dt.strftime(datetime\_output\_format)

# Sort before writing for consistent output

df\_for\_output = df\_for\_output.sort\_values(by='Datetime').reset\_index(drop=True)

df\_for\_output.to\_csv(output\_csv\_path, index=False)

print(f"Processed DataFrame saved to: {output\_csv\_path}")

**From the Command Line**

For quick command-line testing or standalone use, run the script directly:

Bash

python run\_s\_p.py <input\_csv\_file\_path>

**Example:**

Bash

python run\_s\_p.py my\_contest\_log.csv

This will:

1. Load my\_contest\_log.csv into a DataFrame.
2. Apply the Run/S&P inference.
3. Save the result to a new file named my\_contest\_log\_Run.csv (based on the input filename).
4. Print summary statistics to the console.

**Dependencies**

This utility requires the following Python libraries:

* pandas (for DataFrame manipulation)
* Standard Python libraries: collections, os, sys, traceback (these are typically available by default).

You can install pandas using pip:

Bash

pip install pandas

**Known Issues**

* None known in this version (0.9.0-Beta).

**Data Files**

**Input Data Format**

The primary function process\_contest\_log\_for\_run\_s\_p expects a Pandas DataFrame with the following core columns. If loading from a CSV, ensure these columns are present and contain appropriate data:

* **MyCall**: (string) The callsign of the operating station.
* **Datetime**: (string or datetime object) The date and time of the QSO. Expected string format is YYYY-MM-DD HH:MM:SS. The utility will attempt to convert this to a datetime object.
* **Frequency**: (numeric) The operating frequency in Kilohertz (kHz). The utility will convert this to a numeric type.
* **Mode**: (string) The operating mode (e.g., 'CW', 'PH', 'SSB', 'DIG'). The utility differentiates between 'CW' and 'PH' for frequency tolerance.
* **Band**: (string) The operating band (e.g., '160', '80', '40', '20', '15', '10', '6', '2').

Example CSV snippet (header row required):

Code snippet

Contest,Category-Overlay,Category-Operator,Category-Transmitter,Frequency,Mode,Datetime,MyCall,SentClass,SentSection,Call,Class,Section,Band,Dupe,Date,Hour,Prefix,CQZone,ITUZone,Continent,JAContinent,Run,Points,currentscore,currentpoints,currentmult1,currentmult2,Pass

ARRL-FD,,MULTI-OP,UNLIMITED,14025,CW,2025-06-28 18:01:00,W1OP,4A,GA,W4GTA,4A,GA,20,False,2025-06-28,18,K,05,08,NA,NA,Run,Unknown,0,0,0,0,

ARRL-FD,,MULTI-OP,UNLIMITED,14239,PH,2025-06-28 18:02:00,W1OP,4A,GA,K9VQA,1E,IL,20,False,2025-06-28,18,K,05,08,NA,NA,Run,Unknown,0,0,0,0,

ARRL-FD,,MULTI-OP,UNLIMITED,14025,CW,2025-06-28 18:03:00,W1OP,4A,GA,VO1DD,1A,NL,20,False,2025-06-28,18,VE,05,09,NA,NA,S&P,Unknown,0,0,0,0,

ARRL-FD,,MULTI-OP,UNLIMITED,14239,PH,2025-06-28 18:03:00,W1OP,4A,GA,K8NU,1D,OH,20,False,2025-06-28,18,K,05,08,NA,NA,Run,Unknown,0,0,0,0,

**Output Data Format**

The output will be a Pandas DataFrame (and optionally a new CSV file) with all original columns from the input, plus a new column:

* **Run**: (string) The inferred operating style, either 'Run' or 'S&P'.

The output CSV file (e.g., your\_input\_log\_Run.csv) will be sorted by Datetime.